Thwarting Passive Privacy Attacks in Collaborative Filtering

Rui Chen[1], Min Xie[2], and Laks V.S. Lakshmanan[2]

[1] Department of Computer Science, Hong Kong Baptist University
`ruichen@comp.hkbu.edu.hk`
[2] Department of Computer Science, University of British Columbia
`{minxie,laks}@cs.ubc.ca`

Abstract. While recommender systems based on *collaborative filtering* have become an essential tool to help users access items of interest, it has been indicated that collaborative filtering enables an adversary to perform *passive privacy attacks*, a type of the most damaging and easy-to-perform privacy attacks. In a passive privacy attack, the dynamic nature of a recommender system allows an adversary with a moderate amount of background knowledge to infer a user's transaction through temporal changes in the *public* related-item lists (RILs). Unlike the traditional solutions that manipulate the underlying user-item rating matrix, in this paper, we respond to passive privacy attacks by directly anonymizing the RILs, which are the real outputs rendered to an adversary. This fundamental switch allows us to provide a novel rigorous inference-proof privacy guarantee, known as *δ-bound*, with desirable data utility and scalability. We propose anonymization algorithms based on suppression and a novel mechanism, *permutation*, tailored to our problem. Experiments on real-life data demonstrate that our solutions are both effective and efficient.

1 Introduction

In recent years, recommender systems have been increasingly deployed in diverse applications as an effective tool to cope with information overload. Among various approaches developed for recommender systems, *collaborative filtering* (CF) [1] is probably the most successful technique that has been widely adopted. As a standard practice, many CF systems release *related-item lists* (RILs) as a means of engaging users. For example, e-commerce service providers like *Amazon* and *Netflix* have incorporated CF as an essential component to help users find items of interest. Amazon provides RILs as the "Customers who bought this item also bought" feature, while Netflix presents RILs as the "More like" feature. These RILs serve the role of explanations of sorts, which can motivate users to take recommendations seriously.

Though successful as a means of boosting user engagement, it has been recently shown by Calandrino et al. [2] that release of RILs brings substantial privacy risks w.r.t. a fairly simple attack model, known as *passive privacy attack*. In a passive privacy attack, an adversary possesses a moderate amount of *background knowledge* in the form of a subset of items that a *target user* has bought/rated and aims to infer whether a *target item* exists in the target user's transaction. In the sequel, we use the terms *buy* and *rate* interchangeably. The adversary monitors the *public* RIL of each background

	i1	i2	i3	i4	i5	i6	i7	i8
u1	-	2	-	5	▨ 1	-	-	-
u2	3	-	4	-	-	-	-	1
u3	1	-	-	1	3	-	-	-
u4	-	1	-	-	-	2	-	▨ 3
u5	-	3	4	-	2	▨ 5	5	5
u6	2	2	1	-	2	1	3	3
u7	-	▨ 2	-	-	2	-	-	1
u8	-	1	5	-	-	3	-	-

□ Ratings given before time T_1
▨ Ratings given during time $(T_1, T_2]$

(a) A sample user-item rating matrix

Related-item list release at time T_1

i1	i2	i3	i4	i5	i6	i7	i8
i3	i7	i8	i2	i8	i3	i8	i7
i5	i8	i2	i5	i7	i2	i2	i2
i8	i3	i6	i1	i2	i1	i5	i5

Related-item list release at time T_2

i1	i2	i3	i4	i5	i6	i7	i8
i3	i8	i6	i2	i2	i8	i8	i7
i5	i7	i8	i5	i7	i7	i6	i6
i8	i6	i2	i1	i8	i3	i2	i2

(b) Public RILs at different timestamps

Fig. 1. A sample user-item rating matrix and its public RILs

item (i.e., items in the background knowledge) over a period of time. If the target item appears afresh and/or moves up in the RILs of a sufficiently large subset of the background items, the adversary infers that the target item has been added to the target user's transaction. Here is an example that illustrates the idea of passive privacy attacks.

Example 1. Consider a recommender system associated with the user-item rating matrix in Fig. 1(a). Suppose at time T_1 an attacker knows that Alice (user 5) has bought items i_2, i_3, i_7 and i_8 from their daily conversation, and is interested to learn if Alice has bought a sensitive item i_6. The adversary then monitors the temporal changes of the public RILs of i_2, i_3, i_7 and i_8. Let the new ratings made during $(T_1, T_2]$ be the shaded ones in Fig. 1(a). At time T_2, by comparing the RILs with those at T_1, the attacker observes that i_6 appears or moves up in the RILs of i_2, i_3, i_7 and i_8, and consequently infers that Alice has bought i_6. ■

Example 1 demonstrates the possibility of a passive privacy attack. In a real-world recommender system, each change in an RIL is the effect of thousands of transactions. The move-up or appearance of a target item in some background items' RILs may not even be caused by the target user. Thus, one natural question to ask is "*how likely will a passive privacy attack succeed in a real-world recommender system*?". Calandrino et al. [2] perform a comprehensive experimental study on four real-world systems, including *Amazon*, *Hunch*, *LibraryThing* and *Last.fm*, and show that it is possible to infer a target user's unknown transaction with over 90% accuracy on Amazon, Hunch and LibraryThing and 70% accuracy on Last.fm. In particular, passive privacy attacks are able to successfully infer a third of the test users' transactions with *no error* on Hunch. This finding is astonishing as it suggests that the simple passive privacy attack model is surprisingly effective in real-world recommender systems. Therefore there is an urgent need to develop techniques for preventing passive privacy attacks.

Privacy issues in CF have been studied before. With the exception of very few works [3, 4], most proposed solutions [5–11] resort to a distributed paradigm in which user information is kept on local machines and recommendations are generated through the collaboration between a central server and client machines. While this paradigm provides promising privacy guarantees by shielding individual data from the server, it is *not* the current practice of real-world recommender systems. The distributed solution requires substantial architectural changes to existing recommender systems. Worse, the distributed setting does *not* prevent passive privacy attacks because the attacks do not require access to individual user data, but instead rely on aggregate outputs.

Unfortunately, the only works [3, 4] in the centralized setting do not address passive privacy attacks either. Polat and Du [3] suggest to add uniform noise to the user-item rating matrix. However, no formal privacy analysis is provided. In fact, we show in Section 6 that adding uniform noise does not really prevent passive privacy attacks and cannot achieve meaningful utility for RILs. McSherry and Mironov [4] ground their work on *differential privacy* [12], which is known for its rigorous privacy guarantee. They study how to construct a differentially private item covariance matrix, however they do not consider updates to the matrix, an intrinsic characteristic of recommender systems. Furthermore, we argue that differential privacy is *not* suitable for our problem because recent research [13] indicates that differential privacy does *not* provide inferential privacy, which is vital to thwart passive privacy attacks.

Our Contributions. To our best knowledge, *ours is the first remedy to passive privacy attacks in CF, a type of the most damaging and easy-to-perform privacy attacks*. Our contributions are summarized as follows.

First, we analyze the cause of passive privacy attacks, and accordingly propose a novel inference-proof privacy model called *δ-bound* to limit the probability of a successful passive privacy attack. We establish the critical condition for a user-item rating matrix to satisfy δ-bound, which enables effective algorithms for achieving δ-bound.

Second, deviating from the direction of existing studies that manipulates the underlying user-item rating matrix, we address the problem by directly anonymizing RILs. This departure is supported by the fact that, in real-life recommender systems, an adversary does *not* have access to the underlying matrix, and is critical to both data utility and scalability. We propose two anonymization algorithms, one based on suppression and the other based on a novel anonymization mechanism, *permutation*, tailored to our problem. We show that permutation provides better utility.

Third, our anonymization algorithms take into consideration the inherent dynamics of a recommender system. We propose the concept of *attack window* to model a real-world adversary. Our algorithms ensure that the released RILs are private within any attack window in that they satisfy δ-bound w.r.t. passive privacy attacks.

Finally, through an extensive empirical study on real data, we demonstrate that our approach can be seamlessly incorporated into existing recommender systems to provide formal protection against passive privacy attacks while incurring slight utility loss.

2 Related Work

Centralized Private Recommender Systems. There are very few studies on providing privacy protection in centralized recommender systems [3, 4]. Polat and Du [3] suggest users to add uniform noise to their ratings and then send the perturbed ratings to a central recommender system. However, this approach neither provides a formal privacy guarantee and nor prevents passive privacy attacks. McSherry and Mironov [4] show how to generate differentially private item covariance matrices that could be used by the leading algorithms for the Netflix Prize. However, it is *not* known how to apply their approach to a dynamic setting. In contrast, our method aims to support a dynamic recommender system. With a different goal, Machanavajjhala et al. [14] study the privacy-utility tradeoffs in personalized social recommendations. The paper shows that, under differential privacy, it is *not* possible to obtain accurate social recommendations without disclosing sensitive links in a social graph in many real-world settings. These findings motivate us to define a customized privacy model for recommender systems.

Distributed Private Recommender Systems. A large body of research [5–11] resorts to distributed storage and computation of user ratings to protect individual privacy. Canny [5] addresses privacy issues in CF by cryptographic techniques. Users first construct an aggregate model of the user-item rating matrix and then use local computation to get personalized recommendations. Individual privacy is protected by multi-party secure computation. In a later paper [6], Canny proposes a new method based on a probabilistic factor analysis model to achieve better accuracy. Zhang et al. [7] indicate that adding noise with the same perturbation variance allows an adversary to derive significant amount of original information. They propose a two-way communication privacy-preserving scheme, where users perturb their ratings based on the server's guidance. Berkvosky et al. [8] assume that users are connected in a pure decentralized P2P platform and autonomously keep and maintain their ratings in a pure decentralized manner. Users have full control of when and how to expose their data using three general data obfuscation policies. Ahn and Amatriain [10] consider a variant of the traditional CF, known as *expert CF*, in which recommendations are drawn from a pool of domain experts. Li et al. [11] motivate their approach by an active privacy attack model. They propose to identify item-user interest groups and separate users' private interests from their public interests. While this method reduces the chance of privacy attacks, it fails to provide a formal privacy guarantee.

A related research area is *privacy-preserving transaction data publishing* [15] whose goal is to release anonymized transaction databases that satisfy certain privacy models. However, anonymizing the underlying database (e.g., the rating matrix) leads to undesirable data utility and scalability in our problem.

3 Preliminaries

3.1 Item-to-Item Recommendation

A common recommendation model followed by many popular websites is to provide, for every item, a list of its top-N related items, known as *item-to-item recommendation* [2]. Item-to-item recommendations take as input a user-item rating matrix M in which rows correspond to users and columns correspond to items. The set of all users form the user universe, denoted by U; the set of all items form the item universe, denoted by I. Each cell in this matrix represents a user's stated preference (e.g., ratings for movies or historical purchasing information) on an item, and its value is usually within a given range (e.g., $\{1, \cdots, 5\}$) or a special symbol "-", indicating that the preference is unknown. A sample user-item rating matrix is illustrated in Fig. 1(a).

To generate a list of related items for an item i, we calculate *item similarity scores* between i and other items. The similarity scores can be calculated based on some popular approaches, such as *Pearson correlation* and *vector cosine similarity* [16]. The *related item list* (RIL) of an item i is then generated by taking the top-N items that have the largest similarity scores. We call all RILs for all items published at a timestamp T_k an *RIL release*, denoted by $\mathcal{R}_k$. We denote a single RIL of an item j at timestamp T_k by R_k^j. Two sample RIL releases are given in Fig. 1(b).

3.2 Attack Model

In this section, we briefly review passive privacy attacks [2] in CF. In the setting of passive privacy attacks, an adversary possesses some *background knowledge* in the form of a subset of items that have been rated by a *target user*, and seeks to infer whether some other item, called a *target item*, has been rated/bought by the user, from the *public* RIL releases published by the recommender system.

As mentioned in Section 3.1, in item-to-item recommendations, for each item, the recommender system provides an RIL according to item similarity scores. Let an adversary's background knowledge on a target user u_t be B and the target item be $i_t \notin B$. The adversary monitors the changes of the RIL of each *background item* in B over time. If i_t appears afresh and/or moves up in the RILs of a sufficiently large number of background items, indicating the increased similarities between background items and i_t, the adversary might infer that i_t has been added to u_t's transaction, i.e., u_t has bought i_t, with high accuracy.

In reality, an adversary could launch passive privacy attacks by observing the temporal changes between any two RIL releases. However, it is unrealistic to assume that an adversary will perform privacy attacks over an unreasonably long timeframe (e.g., several months or even several years). Therefore, we propose the concept of *attack window* to model a real-world adversary. Without loss of generality, we assume that the RIL releases are generated at consecutive discrete timestamps and an adversary performs attacks at a particular timestamp. We note that this reflects the behavior of real-world recommender systems as RILs are indeed periodically updated. At time T_k, an adversary's attack window $\mathcal{W}_{T_k}$ contains the RIL releases generated at timestamps $T_k, T_{k-1}, \cdots, T_{k-|\mathcal{W}_{T_k}|+1}$, where $|\mathcal{W}_{T_k}|$ is the size of $\mathcal{W}_{T_k}$, namely the number of RIL releases within $\mathcal{W}_{T_k}$. The adversary performs privacy attacks by comparing any two RIL releases within his attack window.

4 Our Privacy Model

To thwart passive privacy attacks in CF, a formal notion of privacy is needed. In the context of *privacy-preserving data publishing*, where an anonymized relational database is published, a plethora of privacy models have been proposed [17]. In contrast, in our problem, recommender systems never publish anonymized rating matrices but only aggregate RILs. In this paper, we propose a novel *inference-proof* privacy notion, known as *δ-bound*, tailored for passive privacy attacks in CF. Let $\mathsf{Tran}(u)$ denote the transaction of user u, i.e., the set of items bought by u.

Definition 1. *(**δ-bound**) Let B be the background knowledge on user u in the form of a subset of items drawn from* $\mathsf{Tran}(u)$*, i.e., $B \subset \mathsf{Tran}(u)$. A recommender system satisfies* δ-bound *with respect to a given attack window $\mathcal{W}$ if by comparing* any *two RIL releases $\mathcal{R}_1$ and $\mathcal{R}_2$ within $\mathcal{W}$, $\max_{u \in U, i \in (I-B)} Pr(i \in \mathsf{Tran}(u) \mid B, \mathcal{R}_1, \mathcal{R}_2) \leq \delta$, where*

$i \in (I - B)$ is any item that either appears afresh or moves up in $\mathcal{R}_2$, and $0 \leq \delta \leq 1$ *is the given privacy requirement.* ■

Intuitively, the definition of δ-bound thwarts passive privacy attacks in item-to-item CF by limiting the probability of a successful attack on *any* user with *any* background items to at most δ. A smaller δ value provides more stringent privacy protection, but may lead to worse data utility. This unveils the fundamental trade-off between privacy and data utility in our problem. We will explore this trade-off in designing our anonymization algorithms in Section 5.

We now analyze the cause of passive privacy attacks and consequently derive the critical condition under which a recommender system enjoys δ-bound. The fundamental cause of passive privacy attacks is that the target user u_t's rating a target item i_t will increase the similarity scores between i_t and the background items B, which might lead to its move-up or appearance in some background items' RILs. So essentially B acts as the *quasi-identifier*, which could potentially be leveraged to identify u_t. u_t's privacy is at risk if B is possessed by only very few users. Consider an extreme example, where u_t is the only user who previously rated B. Suppose that no user who previously rated just part of B rated i_t during the time period $(T_1, T_2]$. Then observing the appearance or move-up of i_t in the RILs of B at T_2 allows the adversary to infer that u_t has rated i_t with 100% probability.

Based on this intuition, one possible way to alleviate passive privacy attacks is to require every piece of background knowledge to be shared by a sufficient number of users. However, this criterion alone is still *not* adequate to ensure δ-bound. Consider an example where, besides u_t, there are another 9 users who also rated B. Suppose, during $(T_1, T_2]$, *all* of them rated i_t. By observing the appearance or move-up of i_t in B's RILs, an adversary's probability of success is still 100%. So, to guarantee δ-bound, it is critical to limit the portion of users who are associated with the background knowledge B and also rated i_t. Let $\mathsf{Sup}(B) \geq 1$ be the number of users u associated with B (i.e., $B \subset \mathsf{Tran}(u)$) at time T_2, $\mathsf{Sup}(B \cup \{i_t\})$ be the number of users who are associated with both B and i_t at T_2. We establish the theorem below.

Theorem 1. *Consider an adversary with background knowledge B on* any *target user u_t. The adversary aims to infer the existence of the target item $i_t \in (I - B)$ in $\mathsf{Tran}(u_t)$ by comparing two RIL releases $\mathcal{R}_1$ and $\mathcal{R}_2$ published at time T_1 and T_2, respectively. If $\frac{\mathsf{Sup}(B \cup i_t)}{\mathsf{Sup}(B)} \leq \delta$, then $Pr(i_t \in \mathsf{Tran}(u_t) \mid B, \mathcal{R}_1, \mathcal{R}_2) \leq \delta$.* ■

Theorem 1 bridges the gap between an attacker's probability of success and the underlying user-item rating matrix, and enables us to guarantee δ-bound by examining the supports in the matrix.

5 Anonymization Algorithms

Achieving δ-bound deals with privacy guarantee. Another equally important aspect of our problem is preserving utility of the RILs. For simplicity of exposition, in this paper we consider the standard utility metric *recall* [18] to measure the quality of anonymized RILs. Essentially, an anonymization algorithm results in better recall if the original RILs and the anonymized RILs contain more common items. It is straightforward to extend our algorithms to other utility metrics.

Our solution employs two *anonymization mechanisms*: *suppression*, a popular mechanism used in privacy-preserving data publishing [17], and *permutation*, a novel mechanism tailored to our problem. Suppression refers to the operation of suppressing an item from an RIL, while permutation refers to the operation of permuting an item that has moved up in an RIL to a position equal to or lower than its original position.

Before elaborating on our algorithms, we give the terminology and notations used in our solution. Recall that an RIL release at timestamp T_k is the set of RILs of all items published at T_k, denoted by $\mathcal{R}_k$. The RIL of an item j at T_k is denoted by R_k^j. Given two timestamps T_1 and T_2 with $T_1 < T_2$ (i.e., T_1 is before T_2), we say that an item i *distinguishes* between R_1^j and R_2^j if one of the following holds: 1) i appears in R_2^j but not in R_1^j, *or* 2) i appears in both R_1^j and R_2^j but its position in R_2^j is higher than its position in R_1^j (i.e., i moves up in R_2^j).

5.1 Suppression-Based Anonymization

Static Release. We start by presenting a simple case, where we are concerned with only two RIL releases (i.e., the attacker's attack window is of size 2). We refer to such a scenario as *static release*. Our goal is to make the second RIL release satisfy δ-bound w.r.t. the first release. We provide an overview of our approach in Algorithm 1.

Algorithm 1. Suppression-based anonymization algorithm for static release

```
Input: User-item rating matrix $M$, RIL release $\mathcal{R}_1$ at time $T_1$, privacy parameter $\delta$
Output: Anonymized RIL release $\mathcal{R}_2$ at time $T_2$
1: Generate $\mathcal{R}_2$ from $M$;
2: for each item $i \in I$ do
3:     Generate the set of items $S_i$ whose RILs are distinguished by $i$;
4: for each item $i \in I$ with $S_i \neq \emptyset$ do
5:     $V_i$ = GenerateViolatingBorder($S_i, \delta, M$)
6:     $L_i$ = IdentifySuppressionLocation($V_i$);
7:     for each location $l \in L_i$ do
8:         Suppress($i, l, M$);
9: return Suppressed $\mathcal{R}_2$;
```

Identify Potential Privacy Threats (Lines 2-3). Since an adversary leverages the temporal changes of the RILs to make inference attacks, the first task is to identify, for each item i, the set of items whose successive RILs at time T_1 and T_2 are distinguished by i. This set of items are referred to as *potential violating items* of i, denoted by S_i. For example, for the two RIL releases in Fig. 1(b), the set of potential violating items of i_6 is $S_{i_6} = \{i_2, i_3, i_7, i_8\}$. An adversary could use any subset of S_i as his background knowledge to infer the existence of i in a target user's transaction.

Determine Suppression Locations (Lines 5-6). Not all these potential violating items (or their combinations, i.e., itemsets) will cause actual privacy threats. Among potential violating items, we identify the itemsets where real privacy threats arise, that is, when an adversary uses the itemsets as his background knowledge, he is able to infer the target item with probability $> \delta$. We eliminate the threats by suppressing the target item from some RILs while achieving minimum utility loss. There are two major technical challenges in doing this, which Algorithm 1 addresses: 1) how to calculate a set of suppression locations s.t. the resultant utility loss is minimized (Lines 5-6); 2) how to suppress an item from an RIL without incurring new privacy threats (Line 8).

For the first challenge, we show that the problem is NP-hard (see Theorem 3 below) and provide an approximation algorithm. For a target item i_t, an adversary's background knowledge could be *any* subset of S_{i_t}. Therefore, we have to guarantee that the probability of inferring the presence of i_t in *any* target user's transaction from *any* itemset $B \subseteq S_{i_t}$, viewed as background knowledge on the target user, is $\leq \delta$. We refer to this probability as the *breach probability* associated with the background knowledge (i.e., itemset) B. We point out that the problem structure does not satisfy any natural monotonicity: indeed, the breach probability associated with an itemset may be more or less than that of its superset. Thus, in the worst case, we must check the breach probability for every itemset (except the empty set) of S_{i_t}, which has exponential complexity. Note that *every* item $i \in I$ could be a target item.

To help tame the complexity of checking all subsets of S_{i_t}, where i_t is *any* candidate target item, we develop a pruning scheme. Define an itemset $s \subset S_{i_t}$ to be a *minimal violating itemset* provided s has a breach probability $> \delta$ and every proper subset of s has a breach probability $\leq \delta$. Let V_{i_t} be the *violating border* of i_t, consisting of *all* minimal violating itemsets of i_t. By definition of minimality, to thwart the privacy attacks on V_{i_t}, it is enough to suppress i_t from the RIL of one item in v, for every minimal violating itemset $v \in V_{i_t}$. The reason is that, for any $v \in V_{i_t}$, no proper subset of v can be used to succeed in an attack. We next show that it is sufficient to guarantee δ-bound on all itemsets in S_{i_t} by ensuring δ-bound on V_{i_t}.

Theorem 2. *For two RIL releases $\mathcal{R}_1$ and $\mathcal{R}_2$, a target user u_t and a target item i_t, $\forall v \in V_{i_t}$, suppressing i_t from the RIL of one item in v ensures $\forall s \subseteq S_{i_t}, Pr(i_t \in \mathsf{Tran}(u_t)|s, \mathcal{R}_1, \mathcal{R}_2) \leq \delta$.* ■

The general idea of $\mathsf{GenerateViolatingBorder}$ (Line 5) is that if an itemset violates δ-bound, then there is no need to further examine its supersets. We impose an arbitrary total order on the items in S_i to ensure that each itemset will be checked exactly once. We iteratively process the itemsets with increasing sizes. The minimal violating itemsets with size k come from a candidate set generated by *joining* non-violating itemsets of size $k-1$. Two non-violating itemsets, $c_1 = \{i_1^1, i_2^1, \cdots, i_l^1\}$ and $c_2 = \{i_1^2, i_2^2, \cdots, i_l^2\}$, can be joined if for all $1 \leq m \leq l-1$, $i_m^1 = i_m^2$ and $\mathsf{Order}(i_l^1) > \mathsf{Order}(i_l^2)$. The joined result is $c_1 \bowtie c_2 = \{i_1^1, i_2^1, \cdots, i_l^1, i_l^2\}$.

For a target item i_t whose potential violating items do not cause any privacy threat, we still need to consider all $2^{|S_{i_t}|} - 1$ itemsets before concluding that there is no threat. To alleviate the computational cost of these items, we make use of a simple pruning strategy. Let the number of users who rated i_t at time T_2 be $\mathsf{Sup}(i_t)$, the number of users rated S_{i_t} at T_2 be $\mathsf{Sup}(S_{i_t})$, and the number of users who rated both S_{i_t} and i_t at T_2 be $\mathsf{Sup}(S_{i_t} \cup \{i_t\})$. Since $\frac{\mathsf{Sup}(S_{i_t} \cup \{i_t\})}{\mathsf{Sup}(S_{i_t})} \leq \frac{\mathsf{Sup}(i_t)}{\mathsf{Sup}(S_{i_t})}$, to guarantee that the breach probability $\frac{\mathsf{Sup}(S_{i_t} \cup \{i_t\})}{\mathsf{Sup}(S_{i_t})} \leq \delta$, it is enough to ensure that $\mathsf{Sup}(S_{i_t}) \geq \frac{\mathsf{Sup}(i_t)}{\delta}$. Notice that, for any subset $s \subset S_{i_t}$, $\frac{\mathsf{Sup}(s \cup \{i_t\})}{\mathsf{Sup}(s)} \leq \frac{\mathsf{Sup}(i_t)}{\mathsf{Sup}(s)} \leq \frac{\mathsf{Sup}(i_t)}{\mathsf{Sup}(S_{i_t})} \leq \delta$. Thus, there is no need to make any checks for subsets of S_{i_t}.

Achieving δ-bound on V_{i_t} requires to find a set of items from whose RILs we suppress the target item i_t, so that after the suppression, for each *potential* background knowledge (i.e., itemset) B, either the breach probability associated with i_t is $\leq \delta$ or i_t does not distinguish the successive RILs of at least one item in B. From a *recall* point of view, we would like to minimize the number of items to be suppressed, since each item suppression leads to a utility loss of 1. This problem can be defined as follows.

Definition 2. *(*IdentifySuppressionLocation*) Given the violating border V_{i_t}, select a set of items L_{i_t} such that $\forall v \in V_{i_t}(\exists l \in L_{i_t}(l \in v))$ and $|L_{i_t}|$ is minimized.* ■

The problem is identical to the *minimal hitting set* (MHS) problem [19], and therefore we have the following theorem.

Theorem 3. IdentifySuppressionLocation *is NP-hard. There is an $O(\ln |V_{i_t}|)$-approximation algorithm to the optimal solution, which runs in O($|V_{i_t}||I|$) time.* ■

Algorithm 2 shows a simple greedy algorithm, which repeatedly picks the item that belongs to the maximum number of "uncovered" itemsets in V_{i_t}, where an itemset is said to be "covered" if one of the items in the current hitting set belongs to it.

Algorithm 2. IdentifySuppressionLocation

```
Input: The violating border $V_i$ of item $i$
Output: A set of locations (items) to suppress $L_i$
1: $L_i = \emptyset$;
2: $C \leftarrow$ the set of items in $V_i$;
3: while $V_i \neq \emptyset$ do
4:     for each item $j \in C$ do
5:         $n_j = |\{v \in V_i : j \in v\}|$;
6:     Add the item $j$ with the maximum $n_j$ to $L_i$;
7:     $V_i = V_i - \{v \in V_i : j \in v\}$
8:     $C \leftarrow$ the set of items in $V_i$;
9: return $L_i$;
```

Perform Suppression (Algorithm 1, Line 8). To thwart privacy attacks, we suppress the target item i_t from the RILs of the items identified by Algorithm 2. Suppressing i_t from a RIL will make items with a position lower than $\mathsf{Pos}(i_t)$ (i.e., the position of i_t in the RIL) move up one position and introduce a new item into the RIL. Note that the move-up or appearance of these items might cause many *new* privacy threats, resulting in both higher complexity and lower utility. To alleviate this problem, instead of changing the positions of all items below $\mathsf{Pos}(i_t)$ and introducing a new item to the RIL, we directly insert a new item at $\mathsf{Pos}(i_t)$ and check its breach probability.

Even inserting a new item i directly at $\mathsf{Pos}(i_t)$ in j's RIL might lead to substantial computational cost, because, in the worst case, it demands to examine every possible combination of the itemsets derived from S_i with j, which is of $O(2^{|S_i|})$ complexity. So we are only interested in items with $S_i = \emptyset$. In this case, we can perform the check in constant time. More specifically, we iteratively consider the items not in the RIL in the descending order of their similarity scores until we find an item to be inserted at $\mathsf{Pos}(i_t)$ without incurring new privacy threats. If an item i not in the RIL has $S_i \neq \emptyset$, we skip i and consider the next item. This process terminates when a qualified item is found. When i is inserted into j's RIL, S_i is accordingly updated: $S_i = \{j\}$.

Multiple Release. We next deal with the case of multiple releases. As discussed in Section 3.2, at any time T_k, an adversary performs passive privacy attacks by comparing any two RIL releases within the attack window $\mathcal{W}_{T_k}$. Hence, whenever a recommender system generates a new RIL release, it has to be secured with respect to all previous $|\mathcal{W}_{T_k}| - 1$ releases. We assume that the attack window size of an adversary is fixed at different timestamps. In reality, this assumption can be satisfied by setting a large enough window size.

We explain the key idea of extending Algorithm 1 for this case. Anonymizing the RIL release $\mathcal{R}_k$ at time T_k works as follows. First, we should generate the potential violating items of *every* item i in $\mathcal{R}_k$ with respect to each of $\mathcal{R}_{k-1}, \mathcal{R}_{k-2}, \cdots, \mathcal{R}_{k-|\mathcal{W}_{T_k}|+1}$. Let $S_i^{\mathcal{R}_j}$ be the potential violating items of i generated by comparing $\mathcal{R}_k$ and $\mathcal{R}_j$, where $k - |\mathcal{W}_{T_k}| + 1 \leq j \leq k-1$. We calculate the violating border over each $S_i^{\mathcal{R}_j}$, denoted by $V_i^{\mathcal{R}_j}$. To make $\mathcal{R}_k$ private for the entire attack window, we need to eliminate all itemsets from these $|\mathcal{W}_{T_k}| - 1$ borders. We take the union of all the borders $V_i = V_i^{\mathcal{R}_{k-1}} \cup \cdots \cup V_i^{\mathcal{R}_{k-|\mathcal{W}|+1}}$. We prune all itemsets that are the supersets of an itemset in V_i, i.e., retain only minimal sets in V_i. The rationale of this pruning step is similar to that of Theorem 2. Second, when we bring in a new item to an RIL, its breach probability needs to be checked with respect to *each* of the previous $|\mathcal{W}_{T_k}| - 1$ releases.

5.2 Permutation-Based Anonymization

In the suppression-based solution, we do not distinguish between an item's appearance and move-up. For items that newly appear in an RIL, we have to suppress them. However, for items that move up in an RIL, we do not really need to suppress them from the RIL to thwart passive privacy attacks. To further improve data utility, we introduce a novel anonymization mechanism tailored to our problem, namely *permutation*. The general idea of permutation is to permute the target item to a lower position in the RIL so that it cannot be used by an adversary to perform a passive privacy attack. If we cannot find a position to permute without generating new privacy threats, we suppress the target item from the RIL. So our permutation-based anonymization algorithm employs both permutation and suppression, but prefers permutation whenever a privacy threat can be eliminated by permutation.

Static Release. Similarly, we first generate the potential violating items S_i for each item i. Unlike in the suppression-based method, we label each item in S_i with either *suppress* or *permute*. If an item gets into S_i due to its appearance in an RIL, it is labeled *suppress*; otherwise it is labeled *permute*. For example, in Fig. 1, we label the occurrences of i_6 in the RILs of i_2, i_7 and i_8 with *suppress* and its occurrence in i_3's RIL with *permute*.

The violating border of S_i can be calculated by the $\mathsf{GenerateViolatingBorder}$ procedure described in Section 5.1. For *recall*, it can be observed that permutation does not incur any utility loss. For this reason, we take into consideration the fact that *suppress* and *permute* are associated with different utility loss when identifying items to anonymize. We call this new procedure IdentifyAnonymizationLocation. We model IdentifyAnonymizationLocation as a *weighted minimum hitting set* (WMHS) problem. IdentifyAnonymizationLocation chooses at every step the item that maximizes the score, namely the ratio between the number of uncovered itemsets containing it and its weight. The weight of an item is calculated based on its utility loss. For an item labeled *suppress*, its weight is 1. For an item labeled *permute*, it does not result in any utility loss and should receive a weight value 0. However, this leads to a divide-by-zero problem. Instead, we assign the item a sufficiently small weight value $\frac{1}{|V_i|+1}$. This is sufficient to guarantee that items labeled *permute* are always preferred over items labeled *suppress*, because the maximum score of an item labeled *suppress* is $|V_i|$ while the minimum score of an item labeled *permute* is $|V_i| + 1$.

To tackle the anonymization locations identified by IdentifyAnonymizationLocation, we start by suppressing items labeled *suppress* because these privacy threats cannot be solved by permutation. Similar to the Suppress procedure described in Section 5.1, we look for the first item i outside an RIL with $S_i = \emptyset$, which does not incur any new privacy threat, as a candidate to replace the suppressed item. One exception is that in the permutation-based solution we can stop searching once we reach the first item that was in the previous RIL (for this type of items there is *no* need to check their breach probability because our following steps make sure that they cannot be used in passive privacy attacks, as is shown later). For the moment, we do not assign a particular position for i and wait for the permutation step. After suppressing all items labeled *suppress*, we perform permutation on the RILs that contain locations returned by IdentifyAnonymizationLocation. In an RIL, for all items that were also in the RIL at the previous timestamp T_1, we assign them the same positions as those at T_1; for all items that were not in the RIL at T_1, we randomly assign them to one of the remaining positions.

We next show the correctness of our permutation-based solution. For an item that needs to be suppressed, it is replaced by a new item, whose appearance is examined to be free of privacy threats, and thus randomly assigning a position does not violate the privacy requirement. For an item that needs to be permuted, we freeze its position to be the same as before, i.e., as in the previous RIL release, and therefore it cannot be used by the adversary to perform passive privacy attacks. So the anonymized RILs are resistant to passive privacy attacks.

Multiple Release. Finally, we explain our permutation-based algorithm for the *multiple release* scenario. Algorithm 3 presents our idea in detail. We compare the true $\mathcal{R}_k$ at time T_k with each of the previous $|\mathcal{W}_{T_k}| - 1$ RIL releases within the attack window $\mathcal{W}_{T_k}$ to generate the corresponding potential violating items for each item i, denoted by $S_i^{\mathcal{R}_j}$ (Lines 2-4). In addition to labeling each potential violating item by *suppress* or *permute*, for an item i labeled *permute*, we record its position in the RIL in which it moves up (Line 5).

For each $S_i^{\mathcal{R}_j}$, we calculate its violating border $V_i^{\mathcal{R}_j}$ (Line 6). Since we have to make $\mathcal{R}_k$ private with respect to all previous $|\mathcal{W}_{T_k}| - 1$ releases, we perform a union

Algorithm 3. Permutation-based anonymization algorithm for multiple release

```
Input: User-item rating matrix $M$, the attack window $\mathcal{W}_{T_k}$ at time $T_k$, previous $|\mathcal{W}_{T_k}| - 1$ RIL releases, privacy parameter $\delta$
Output: Anonymized RIL release $\mathcal{R}_k$ at time $T_k$
 1: Generate $\mathcal{R}_k$ from $M$;
 2: for each previous RIL release $\mathcal{R}_j$ do
 3:     for each item $i \in I$ do
 4:         Generate the set of items $S_i^{\mathcal{R}_j}$ whose RILs are distinguished by $i$ between $\mathcal{R}_k$ and $\mathcal{R}_j$;
 5:         Label items in $S_i^{\mathcal{R}_j}$ by $suppress$ or $permute$ and record $permute$ position;
 6:         $V_i^{\mathcal{R}_j} = \mathsf{GenerateViolatingBorder}(S_i^{\mathcal{R}_j}, \delta, M)$;
 7: for each item $i \in I$ do
 8:     $V_i = V_i^{\mathcal{R}_{k-1}} \cup \cdots \cup V_i^{\mathcal{R}_{k-|\mathcal{W}_{T_k}|+1}}$;
 9:     $V_i = \mathsf{Label}(V_i)$;
10:     $V_i = \mathsf{Prune}(V_i)$;
11:     for each item $i \in I$ with $V_i \neq \emptyset$ do
12:         $\langle L_i, C_i \rangle = \mathsf{IdentifyAnonymizationLocation}(V_i)$;
13:         for each location-code pair $\langle l, c \rangle \in \langle L_i, C_i \rangle$ do
14:             if $c = suppress$ then
15:                 $\mathsf{SuppressMR}(i, l, M)$;
16:             else
17:                 $\mathsf{PermuteMR}(i, l, M)$;
18: return Anonymized $\mathcal{R}_k$;
```

over all $V_i^{\mathcal{R}_j}$ (Line 8). In the case of multiple release, the same item might be labeled both *suppress* and *permute* in different $V_i^{\mathcal{R}_j}$ and by different positions. To resolve this inconsistency, we let *suppress* take precedence over *permutation*. That is, if an item i is labeled *suppress* in any $V_i^{\mathcal{R}_j}$, it will be labeled *suppress* in V_i (Line 9), because a new item's entering in an RIL cannot be hidden by permuting its position. Also, the position associated with an item labeled *permute* is updated to the lowest position of all its positions in different $V_i^{\mathcal{R}_j}$. We call this lowest position the *safe position*. It is not hard to see that only if the item is permuted to a position lower than or equal to its safe position, it can be immune to passive privacy attacks within the entire attack window. A similar pruning strategy can be applied on V_i, which removes all supersets of an itemset in V_i (Line 10).

V_i is then fed into IdentifyAnonymizationLocation (Line 12). The outputs are a set of items (i.e., locations) in whose RIL i should be anonymized, their corresponding anonymization codes (either *suppress* or *permute*), and safe positions for items labeled *permute*. For items labeled *suppress*, they are processed with the same procedure as the suppression-based solution for the multiple release scenario (SuppressMR). Here we focus on PermuteMR (Line 17). In static release, we can restore the items labeled *permute* to their previous positions to thwart privacy attacks. However, this is not sufficient for multiple release, because changes of the underlying user-item rating matrix are different in different time periods.

The key observation is that we have to permute the target item i_t to a position lower than or equal to its safe position. We iteratively switch i_t with the items in the RIL with position lower than or equal to i_t's safe position and check if the switch incurs any new privacy threat with respect to all previous $|\mathcal{W}_{T_k}| - 1$ releases. If we cannot find a permutation without violating the privacy requirement, we suppress i_t instead.

6 Experiments

In this section, we study the performance of the proposed anonymization algorithms over the public real-life datasets *MovieLens* and *Flixster*. We compare our suppression-based anonymization algorithm (*SUPP*) and permutation-based anonymization algorithm (*PERM*) with the randomized perturbation approach (*RP*) [3]. Due to the reason explained in Section 2, we cannot perform a meaningful comparison with the approach in [4]. All implementations were done in Python, and all experiments were run on a Linux machine with a 4 Core Intel Xeon CPU and 16GB of RAM.

The objectives of our experiments are: 1) evaluate the utility of various anonymization algorithms under different parameters; 2) examine the probability of successful passive privacy attacks after performing different anonymization algorithms; and 3) demonstrate the efficiency of our proposed algorithms.

The first dataset *MovieLens* (http://www.movielens.org) is a popular recommendation benchmark. It contains 1 million ratings over 4K movies and 6K users. The second dataset *Flixster* was crawled from the Flixster website [20], and contains 8.4 million ratings over 49K movies and 1 million users. Both datasets are time-stamped, and in all experiments, we follow the classical item-based recommendation framework studied in [18] to calculate item similarity scores. For *RP*, we use zero-meaned uniform noise with small variances. Experimental results obtained under different variances exhibit similar trends. Due to space limit, we only report the results with the variance equal to 1.

For all experiments, we select the initial timestamp such that the initial RIL release is generated based on approximately 10% of all ratings in the dataset. For the time gap between two consecutive RIL releases, we consider it to be a time period for generating a multiple of 1% of total ratings, e.g., if the time gap is 5, then the number of ratings generated between the two consecutive RIL releases will be approximately 5% of all ratings. Results obtained from other settings of these two parameters are very similar, and hence omitted here. In all experiments, we consider the effect of four tunable parameters: the attack window size, the time gap between two consecutive RIL releases, the privacy requirement δ, and the number of items in an RIL N. The following default values are used unless otherwise specified: 4 for the attack window size, 5 for the time gap, 0.1 for δ, and 5 for N.

Utility Study. As discussed before, *SUPP* and *PERM* only anonymize a few RILs in which real privacy risks for passive privacy attacks arise. Thus, they will leave most of the RILs intact. This is confirmed by *overall recall*, which is defined as the percentage of items in *all* original RILs that are retained after the anonymization. We show in Fig. 2 the overall recall of different algorithms on both datasets by varying the four parameters, namely attack window size, time gap between two consecutive RIL releases, δ, and N. It can be observed that both *SUPP* and *PERM* consistently achieve high overall recall, while *RP* cannot provide desirable utility in terms of RILs.

To further examine the utility loss just on the anonymized RILs (by ignoring RILs which are intact after the anonymization), we also consider *targeted recall*, which is

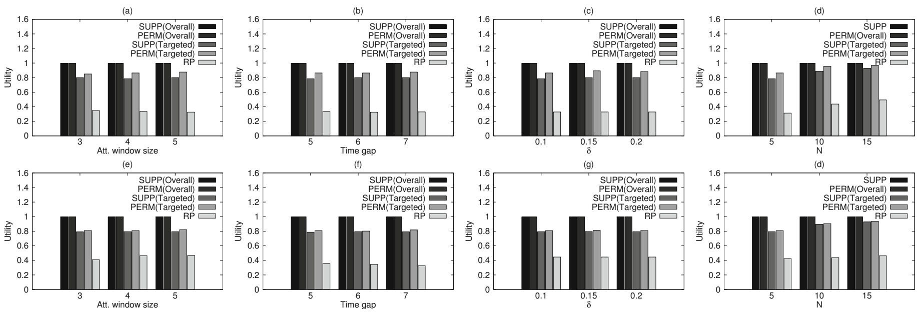

Fig. 2. Utility results on: MovieLens (a)–(d); Flixster (e)–(h)

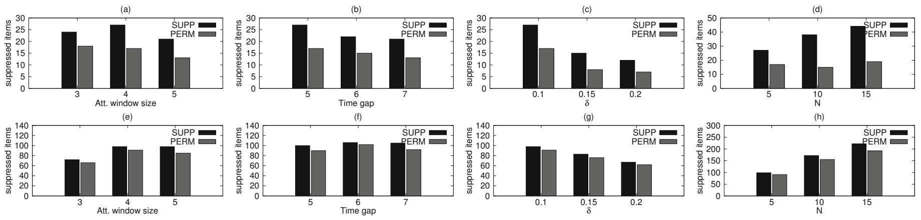

Fig. 3. Number of items suppressed by different algorithms: MovieLens (a)–(d); Flixster (e)–(h)

defined as the percentage of items retained in the anonymized RILs (i.e., the RILs in which suppression and/or permutation are performed). This utility metric is of importance because we do not want to have anonymized RILs that are substantially different from the original ones. The experimental results on both datasets, as shown in Fig. 2, suggest that our algorithms do not significantly destroy the usefulness of any RIL. We can also observe that *PERM* achieves better utility than *SUPP*. We present the numbers of suppressed items by both *PERM* and *SUPP* under varying parameters in Fig. 3. The results confirm that *PERM* is more preferable than *SUPP* in all cases.

Privacy Study. In Fig. 4, for both datasets, we demonstrate that *RP* cannot prevent passive privacy attacks: the worst case breach probability over the RILs generated from the perturbed user-item rating matrix is still extremely high (e.g., 100% for some target user). In contrast, our algorithms ensure that the breach probability over anonymized RILs is always less than the given privacy parameter δ.

Efficiency Study. Finally, we show the run-time of our proposed anonymization algorithms under various settings over both datasets in Fig. 5. As can be observed, both proposed algorithms are efficient, and in most situations, *PERM* is at least twice as fast as *SUPP*. The reason is that the cost of permutation is often much smaller than suppression, since the latter may need to explore many items beyond an RIL before finding a qualified replacement. Therefore, we conclude that empirically *PERM* is a better choice than *SUPP* in terms of both utility and efficiency.

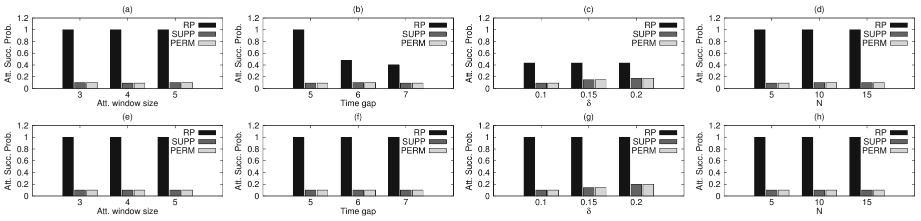

Fig. 4. Attack success probability results on: MovieLens (a)–(d); Flixster (e)–(h)

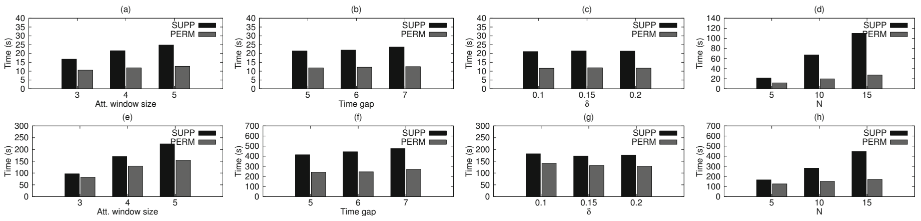

Fig. 5. Efficiency results on: MovieLens (a)–(d); Flixster (e)–(h)

7 Conclusion

The recent discovery of passive privacy attacks in item-to-item CF has exposed many real-life recommender systems to a serious compromise of privacy. In this paper, we propose a novel inference-proof privacy notion called δ-*bound* for thwarting passive privacy attacks. We develop anonymization algorithms to achieve δ-*bound* by means of a novel anonymization mechanism called *permutation*. Our solution can be seamlessly incorporated into existing recommender systems as a post-processing step over the RILs generated using traditional CF algorithms. Experimental results demonstrate that our solution maintains high utility and scales to large real-life data.

References

1. Goldberg, D., Nichols, D., Oki, B.M., Terry, D.: Using collaborative filtering to weave an information tapestry. Communications of ACM 35(12), 61–70 (1992)
2. Calandrino, J.A., Kilzer, A., Narayanan, A., Felten, E.W., Shmatikov, V.: "You might also like": Privacy risks of collaborative filtering. In: S&P (2011)
3. Polat, H., Du, W.: Privacy-preserving collaborative filtering using randomized perturbation techniques. In: ICDM (2003)
4. McSherry, F., Mironov, I.: Differentially private recommender systems: building privacy into the Netflix prize contenders. In: SIGKDD (2009)
5. Canny, J.: Collaborative filtering with privacy. In: S&P (2002)
6. Canny, J.: Collaborative filtering with privacy via factor analysis. In: SIGIR (2002)
7. Zhang, S., Ford, J., Makedon, F.: A privacy-preserving collaborative filtering scheme with two-way communication. In: EC (2006)

8. Berkvosky, S., Eytani, Y., Kuflik, T., Ricci, F.: Enhancing privacy and preserving accuracy of a distributed collaborative filtering. In: RecSys (2007)
9. Aimeur, E., Brassard, G., Fernandez, J.M., Onana, F.S.M.: ALAMBIC: A privacy-preserving recommender system for electronic commerce. International Journal of Information Security 7(5), 307–334 (2008)
10. Ahn, J.W., Amatriain, X.: Towards fully distributed and privacy-preserving recommendations via expert collaborative filtering and restful linked data. In: WI-IAT (2010)
11. Li, D., Lv, Q., Xia, H., Shang, L., Lu, T., Gu, N.: Pistis: A privacy-preserving content recommender system for online social communities. In: WI-IAT (2011)
12. Dwork, C., McSherry, F., Nissim, K., Smith, A.: Calibrating noise to sensitivity in private data analysis. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 265–284. Springer, Heidelberg (2006)
13. Cormode, G.: Personal privacy vs population privacy: Learning to attack anonymization. In: SIGKDD (2011)
14. Machanavajjhala, A., Korolova, A., Sarma, A.D.: Personalized social recommendations: Accurate or private. PVLDB 4(7), 440–450 (2011)
15. Ghinita, G., Tao, Y., Kalnis, P.: On the anonymization of sparse high-dimensional data. In: ICDE (2008)
16. Su, X., Khoshgoftaar, T.M.: A survey of collaborative filtering techniques. Advances in Artificial Intelligence (2009)
17. Fung, B.C.M., Wang, K., Chen, R., Yu, P.S.: Privacy-preserving data publishing: A survey of recent developments. ACM Computing Surveys 42(4), 14:1–14:53 (2010)
18. Karypis, G.: Evaluation of item-based top-n recommendation algorithms. In: CIKM (2001)
19. Ausiello, G., D'Atri, A., Protasi, M.: Structure preserving reductions among convex optimization problems. Journal of Computer and System Sciences 21(1), 61–70 (1980)
20. Jamali, M., Ester, M.: A matrix factorization technique with trust propagation for recommendation in social networks. In: RecSys (2010)